Fine Art Search Engine

Rutuja Gurav, Krupa Hegde and Gautham Mani

Department of Computer Science, University of California, Riverside, CA, USA
{rgura001@ucr.edu, khegd001@ucr.edu, gmani001@ucr.edu}

1 Introduction

On the web there are several online stores to buy various kinds of Fine Art. Fine Arts aficionados visit a number of different websites before purchasing a painting. We sought to provide an one-stop interface to art lover for browsing paintings for sale across various online websites.

In this project, we are building a search engine for Fine Art on sale on the web. We chose 5 online art stores selling art ranging from professional retailers to amateur artists. We crawled these websites to collect the data on paintings for sale and created an index to rank and search paintings and present the user with top results from across these websites.

Section 2 gives the collaboration details of the members who have contributed to this project. Section 3 describes the crawling system and highlights the crawling strategy used. Section 4 tells about the limitations of this projects and the obstacles that we faced. Section 5 gives an overview of Lucene indexing strategy. Section 6 shows the runtime statistics of the Lucene index creation process. And Section 7 contains instructions for deploying the crawler and building the index.

2 Collaborations Details

1. Name: Gautam Mani
   Contribution Details:
   1. Brainstormed for project ideas.
   2. Built a crawler for website viz. fineartsamerica.com
   3. Used Lucene API to build the index.
2. Name: Krupa Hegde
   Contribution Details:
   1. Brainstormed for project ideas.
   2. Built crawlers for 2 websites viz. SaatchiArt.com, Etsy.com
   3. Used Lucene API to build the index.
3. Name: Rutuja Gurav
   Contribution Details:
   1. Brainstormed for project ideas.
   2. Built crawlers for 2 websites viz. Deviantart.com, Fizdi.com
   3. Image processing to extract dominant colors in the painting images.

3 Overview of crawling system

3.1 Architecture

A web crawler is an automated script or program for scraping information from webpages for purpose of analysis, indexing, searching and querying.A typical architecture of a web crawler is as follows

The source of data to be crawled is the web from where a pool of seed pages is selected. In our case, the 5 websites which we selected are SaatchiArt.com, Etsy.com, DeviantArt.com, FineArtsAmerica.com and Fizdi.com.

The crawler program then scraps all necessary information from the seed pages and enqueues the URLs of the links found in the seed pages in a queue. Once the crawler is finished scarping the current page, it stores the data to file and it dequeues a page from the front of the queue to begin scrapping again. This process continues in this non-terminating fashion unless we provide some limit to total number of pages to be crawled.

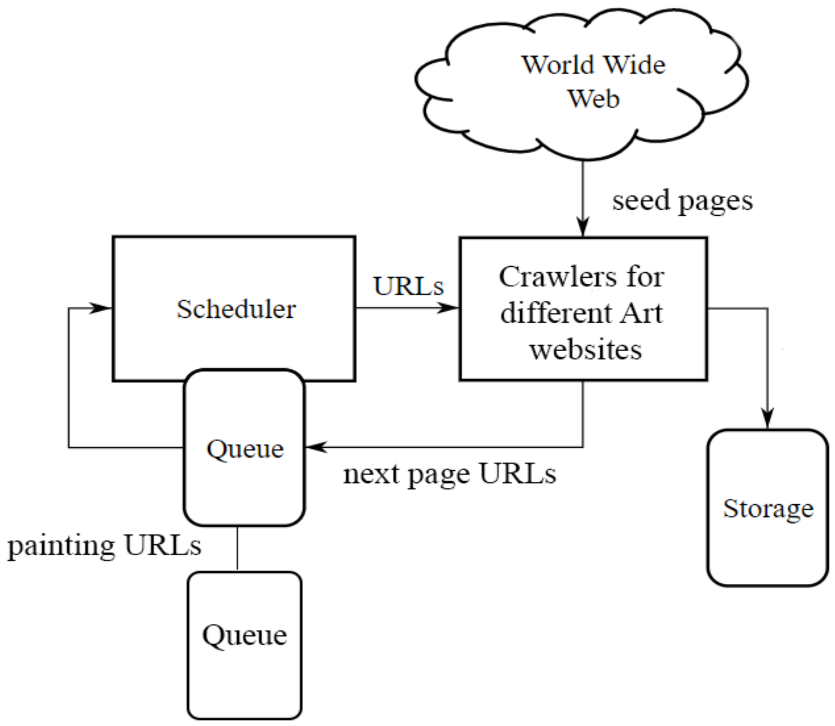


Fig. 1. Architecture

3.2 The Crawling Strategy

We have built 5 parsers for crawlers for our 5 seed websites as their DOM structures are different from each other. The websites display approximately 24 to 70 thumbnails of painting for sale accompanied by URLs to go to an individual paintings page.

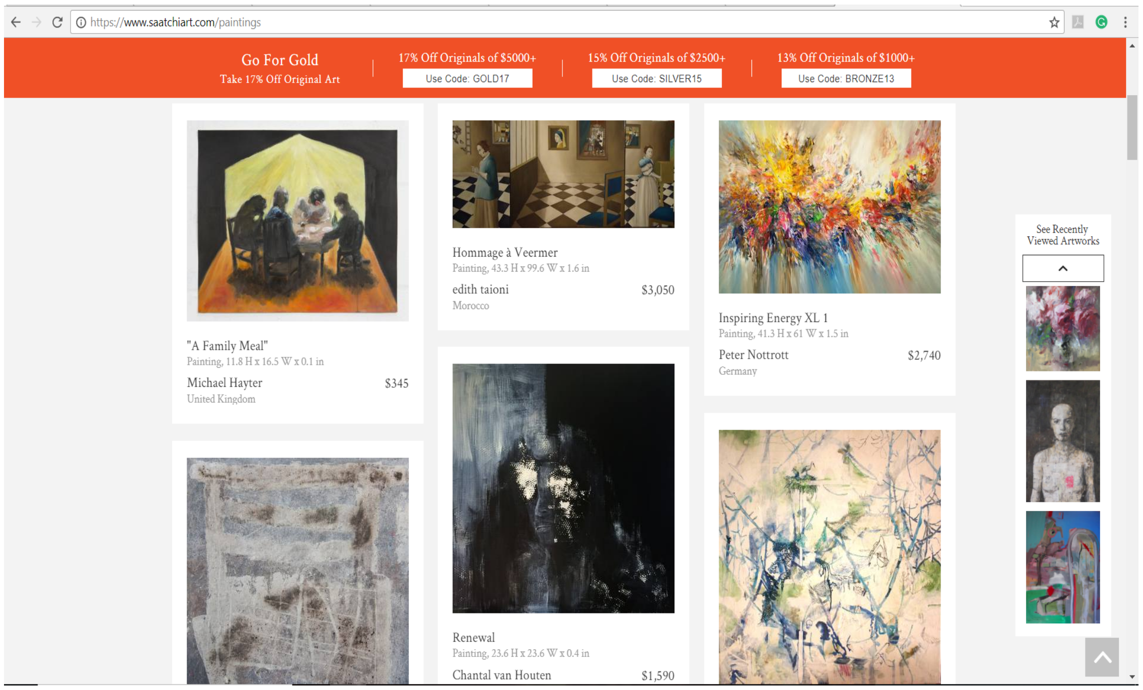
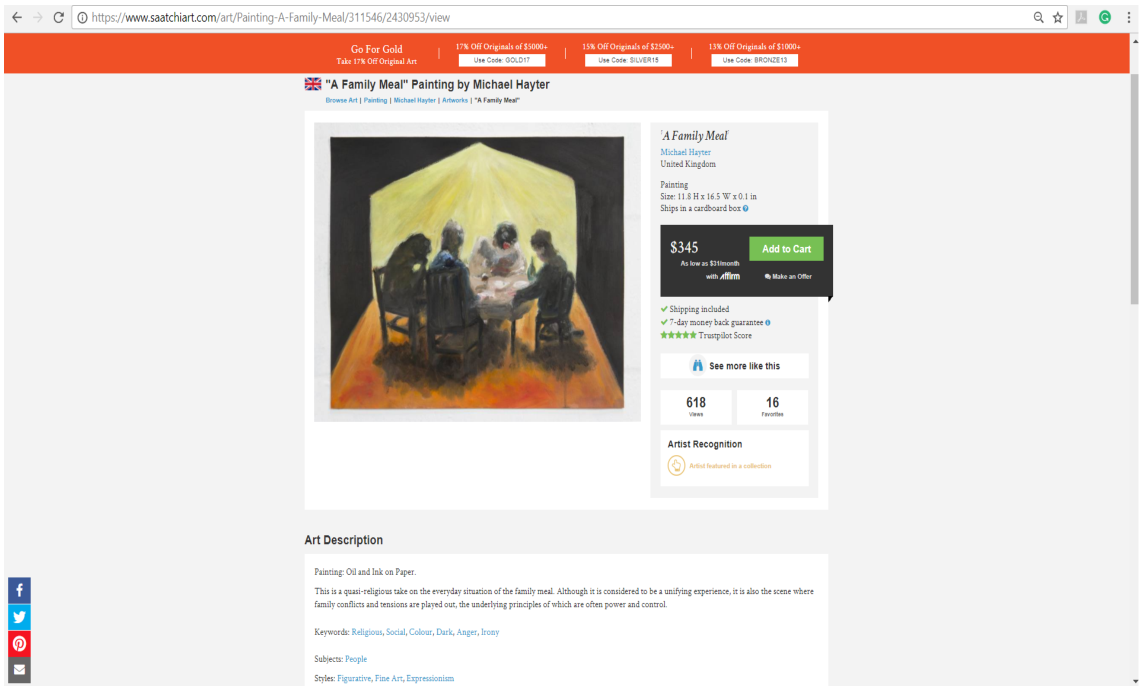

https://www.saatchiart.com/paintings
Go For Gold
Take 17% Off Original Art
17% Off Originals of $5000+
Use Code: GOLD17
15% Off Originals of $2500+
Use Code: SILVER15
13% Off Originals of $1000+
Use Code: BRONZE13
"A Family Meal"
Painting, 11.8 H x 16.5 W x 0.1 in
Michael Hayter $345
United Kingdom
Hommage à Veermer
Painting, 43.3 H x 99.6 W x 1.6 in
edith taioni $3,050
Morocco
Inspiring Energy XL 1
Painting, 41.3 H x 61 W x 1.5 in
Peter Nottrott $2,740
Germany
Renewal
Painting, 23.6 H x 23.6 W x 0.4 in
Chantal van Houten $1,590
See Recently Viewed Artworks

Fig. 2. Main page

We have implemented a breadth-first search crawling strategy for this project. The crawler enqueues all the painting URLs from the main page into a links queue. Once all the painting URLs from a main page are added to the queue, the crawler then visits individual painting pages and parses the HTML source to scrap information about the painting such as painting title, artist, description, medium, tags, size, prices, views, likes.

We have also downloaded the image to perform image processing in order to extract dominant colors from the paintings. We incorporated these colors as tags for richer indexing.We have crawled 5.8 GB data with around 70,000 items each image contributing 10-200 kb.

Duplicate elimination The websites which we have crawled tended to dynamically change as a next page is loaded to be crawled. So, in order to eliminate


https://www.saatchiart.com/art/Painting-A-Family-Meal/311546/2430953/view
Add to Cart
Make an Offer
See more like this

Go For Gold
Take 17% Off Original Art
17% Off Originals of $5000+ Use Code: GOLD17
15% Off Originals of $2500+ Use Code: SILVER15
13% Off Originals of $1000+ Use Code: BRONZE13

"A Family Meal" Painting by Michael Hayter

Browse Art | Painting | Michael Hayter | Artworks | "A Family Meal"

'A Family Meal'
Michael Hayter
United Kingdom

Painting
Size: 11.8 H x 16.5 W x 0.1 in
Ships in a cardboard box

$345
As low as $31/month with Affirm

✓ Shipping included
✓ 7-day money back guarantee
★★★★★ Trustpilot Score

618 Views
16 Favorites

Artist Recognition
Artist featured in a collection

Art Description

Painting: Oil and Ink on Paper.

This is a quasi-religious take on the everyday situation of the family meal. Although it is considered to be a unifying experience, it is also the scene where family conflicts and tensions are played out, the underlying principles of which are often power and control.

Keywords: Religious, Social, Colour, Dark, Anger, Irony

Subjects: People

Styles: Figurative, Fine Art, Expressionism

Fig. 3. Painting page

duplicates from crawled painting URLs, we hashed the painting URLs such that there would be no duplicate data scrapped and stored in the JSONs by making use of HashSet collection in Java.

Crawling Policies We adhered to the gentlemans agreement of not inundating the website servers with too many requests per second and scheduled our threads properly with sufficient delays.

4 Limitations and Obstacles

Due to the nature of the domain that we chose, we were unable to fully parallelize the crawling process as we were scrapping 5 different websites with different DOM structures. Since the request were to the same website again and again and we had to maintain delays we did not create new threads for each new urls. Instead, we multi-threaded the crawling process for the 5 websites by dedicating one thread per website.

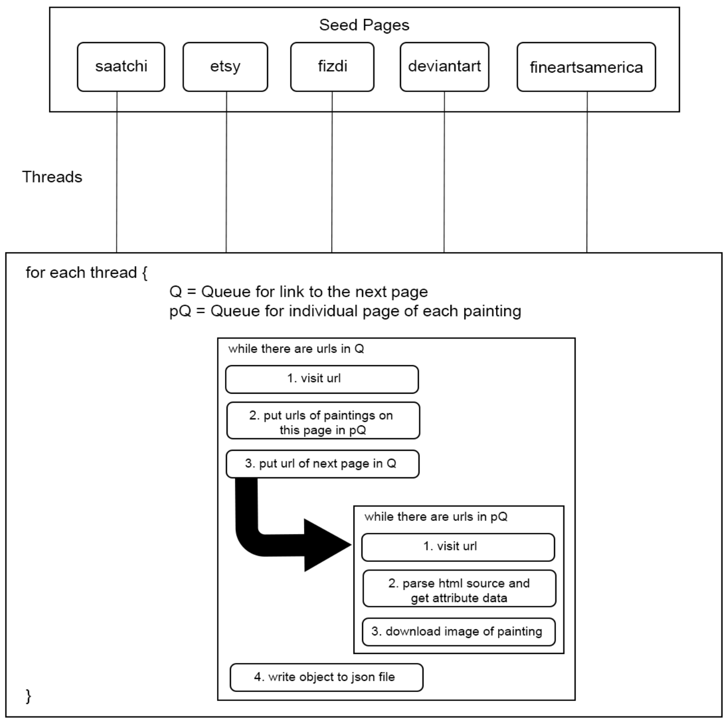

Seed Pages
saatchi
etsy
fizdi
deviantart
fineartsamerica
Threads
for each thread {
Q = Queue for link to the next page
pQ = Queue for individual page of each painting
while there are urls in Q
1. visit url
2. put urls of paintings on this page in pQ
3. put url of next page in Q
while there are urls in pQ
1. visit url
2. parse html source and get attribute data
3. download image of painting
4. write object to json file
}

Fig. 4. Crawler

5 Overview of Lucene indexing strategy

5.1 Fields in the Lucene index

Fields	Field Type	Justification
URL	StringField	We want it to be stored and indexed but not tokenized.
Description	TextField	Tokenized and indexed.
Medium	TextField	Tokenized and indexed.
Title	TextField	Tokenized and indexed.
Color	TextField	JSON array of size 2, contains two dominant colors from the painting image. Tokenized and indexed.
Tags	TextField	JSON array containing individual tags as values. Tokenized and indexed.
Artist	TextField	Tokenized and indexed.
Artist URL	StoredField	Stored, but not indexed or tokenized
Size	StoredField	Stored, but not indexed or tokenized
Price	StoredField	Stored, but not indexed or tokenized
Views	StoredField	Stored, but not indexed or tokenized
Likes	StoredField	Stored, but not indexed or tokenized
Image Source	StoredField	Stored, but not indexed or tokenized

Text analyzer choices The StandardAnalyser has a StandardTokenizer, StandardFilter, LowercaseFilter, and StopFilter.

We examined EnglishAnalyser which has some English stemming enhancements which work well for plain English text. However, since we are building an image search engine essentially based on tags and short descriptions, we do not have extensive text which would require such enhancements.

Standard Analyzer tokenizes cleverly the following lexical types: alphanumerics, acronyms, company names, e-mail addresses, computer host names, numbers, words with an interior apostrophe, serial numbers, IP addresses, and CJK (Chinese Japanese Korean) characters. StandardAnalyzer also includes stop-word removal, using the same mechanism as the StopAnalyzer which is sufficient for the purposes of our project.

We have not provided Analyzer options as an input argument while building the index because we are using only StandardAnalyzer by default.

6 Runtime of Lucene index creation process

The runtime statistics of index creation are shown in the figure below -

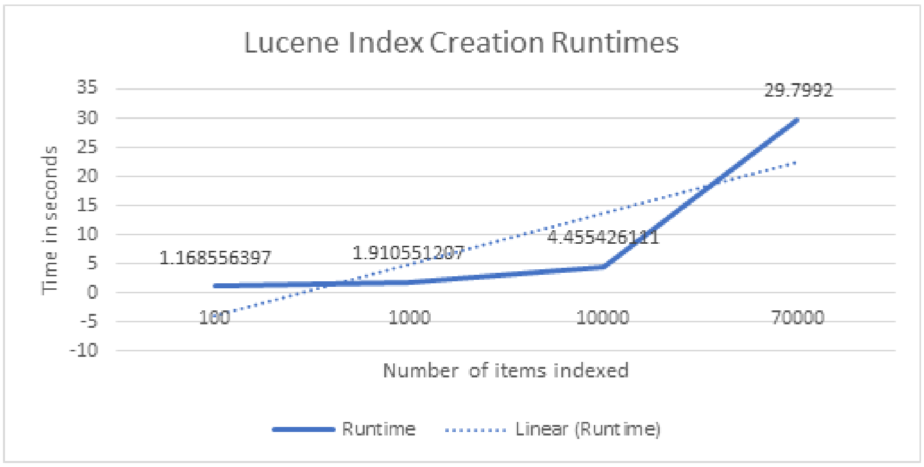


Fig. 5. Runtime Statistics

The trendline shows that the runtime steadily increases as the number of items indexed increase.

7 Instruction on how to deploy the crawler and build index

1. Extract the zip file which contains.
   a. startCrawler.sh
   b. buildIndex.sh
   c. jar files for crawler and indexer
   d. OtherFiles folder contains Java codes for Crawlers and Lucene Index Creation. It also contains a zip file of codes for image processing getColorAttribute.zip
   (These are MATLAB codes which wont be executed in the flow of this execution. In case you want to run them, please use MATLAB and consult the README.)
2. From the folder path, run the following command in terminal to start the crawlers
   $./startcrawler.sh < seed.txtfilepath >< no.ofpagestobecrawled >< output_directory >$
   Example: $./startCrawler.shseed.txt100/home/user/output$
   NOTE: The seed URLs in the seed.txt must be in the same order as provided because we are reading seed.txt and creating threads in that order.
3. Once the crawlers terminate and JSON files are created, run the following command to build the Lucene Index.
   $./buildIndex.sh < output_directory >< input_directory >$
   NOTE: The input directory for the index building process must be a directory containing only the JSON files and not a file or a directory containing other types of files.

References

1. Cookbook:JSoup Java HTML parser: https://jsoup.org/cookbook/
2. Apache Lucene Guide: https://lucene.apache.org/
3. MatLab Guide for Image Processing: https://www.mathworks.com/help/images/getting-started-with-image-processing-toolbox.html
4. Book by Erik Hatcher and Otis Gospodneti (2004): Lucene in Action
5. General Syntaxes: https://stackoverflow.com/
6. Json-simple: https://code.google.com/archive/p/json-simple/